Logical Methods in Computer Science
Vol. 9(1:09)2013, pp. 1–27
www.lmcs-online.org

Submitted Jun. 16, 2011
Published Mar. 4, 2013

GENERALIZING DETERMINIZATION FROM AUTOMATA TO COALGEBRAS

ALEXANDRA SILVA [a], FILIPPO BONCHI [b], MARCELLO BONSANGUE [c], AND JAN RUTTEN [d]

[a] Radboud University Nijmegen and Centrum Wiskunde & Informatica
e-mail address: ams@cwi.nl

[b] ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)
e-mail address: filippo.bonchi@ens-lyon.fr

[c] LIACS - Leiden University
e-mail address: marcello@liacs.nl

[d] Centrum Wiskunde & Informatica and Radboud University Nijmegen
e-mail address: janr@cwi.nl

Abstract. The powerset construction is a standard method for converting a nondeterministic automaton into a deterministic one recognizing the same language. In this paper, we lift the powerset construction from automata to the more general framework of coalgebras with structured state spaces. Coalgebra is an abstract framework for the uniform study of different kinds of dynamical systems. An endofunctor F determines both the type of systems (F-coalgebras) and a notion of behavioural equivalence ($\sim_F$) amongst them. Many types of transition systems and their equivalences can be captured by a functor F. For example, for deterministic automata the derived equivalence is language equivalence, while for non-deterministic automata it is ordinary bisimilarity.

We give several examples of applications of our generalized determinization construction, including partial Mealy machines, (structured) Moore automata, Rabin probabilistic automata, and, somewhat surprisingly, even pushdown automata. To further witness the generality of the approach we show how to characterize coalgebraically several equivalences which have been object of interest in the concurrency community, such as failure or ready semantics.

2012 ACM CCS: [**Theory of computation**]: Models of computation—Abstract machines & Formal languages and automata theory—Formalisms—Algebraic language theory & Semantics and reasoning—Program semantics—Categorical semantics.

Key words and phrases: Coalgebras, Powerset Construction, Linear Semantics.

[a] The work of Alexandra Silva is partially funded by the ERDF through the Programme COMPETE and by the Portuguese Foundation for Science and Technology, project ref. `PTDC/EIA-CCO/122240/2010` and `SFRH/BPD/71956/2010`.

[b] The work of Filippo Bonchi is supported by the CNRS PEPS project CoGIP and the project ANR 12IS02001 PACE.

[c,d] The research of Marcello Bonsangue and Jan Rutten has been carried out under the Dutch NWO project *CoRE: Coinductive Calculi for Regular Expressions.*, dossier number 612.063.920.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-9(1:09)2013

Introduction

Coalgebra is by now a well established general framework for the study of the behaviour of large classes of dynamical systems, including various kinds of automata (deterministic, probabilistic etc.) and infinite data types (streams, trees and the like). For a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, an F-coalgebra is a pair (X, f), consisting of a set X of states and a function $f\colon X \to F(X)$ defining the observations and transitions of the states. Coalgebras generally come equipped with a standard notion of equivalence called *F-behavioural equivalence* that is fully determined by their (functor) type F. Moreover, for most functors F there exists a *final* coalgebra into which any F-coalgebra is mapped by a unique homomorphism that identifies all F-equivalent states.

Much of the coalgebraic approach can be nicely illustrated with deterministic automata (DA), which are coalgebras of the functor $D(X) = 2 \times X^A$. In a DA, two states are D-equivalent precisely when they accept the same language. The set 2^{A^*} of all formal languages constitutes a final D-coalgebra, into which every DA is mapped by a homomorphism that sends any state to the language it accepts.

It is well-known that *non-deterministic* automata (NDA) often provide more efficient (smaller) representations of formal languages than DA's. Language acceptance of NDA's is typically defined by turning them into DA's via the *powerset construction*. Coalgebraically this works as follows. NDA's are coalgebras of the functor $N(X) = 2 \times \mathcal{P}_\omega(X)^A$, where $\mathcal{P}_\omega$ is the finite powerset. An N-coalgebra $(X, f\colon X \to 2 \times \mathcal{P}_\omega(X)^A)$ is *determinized* by transforming it into a D-coalgebra $(\mathcal{P}_\omega(X), f^\sharp\colon \mathcal{P}_\omega(X) \to 2 \times \mathcal{P}_\omega(X)^A)$ (for details see Section 2). Then, the language accepted by a state s in the NDA (X, f) is defined as the language accepted by the state $\{s\}$ in the DA $(\mathcal{P}_\omega(X), f^\sharp)$.

For a second variation on DA's, we look at *partial automata* (PA): coalgebras of the functor $P(X) = 2 \times (1 + X)^A$, where for certain input letters transitions may be undefined. Again, one is often interested in the DA-behaviour (i.e., language acceptance) of PA's. This can be obtained by turning them into DA's using *totalization*. Coalgebraically, this amounts to the transformation of a P-coalgebra $(X, f\colon X \to 2\times(1+X)^A)$ into a D-coalgebra $(1 + X, f^\sharp\colon 1 + X \to 2 \times (1 + X)^A)$.

Although the two examples above may seem very different, they are both instances of one and the same phenomenon, which it is the goal of the present paper to describe at a general level. Both with NDA's and PA's, two things happen at the same time: (i) more (or, more generally, different types of) transitions are allowed, as a consequence of changing the functor type by replacing X by $\mathcal{P}_\omega(X)$ and $(1+X)$, respectively; and (ii) the behaviour of NDA's and PA's is still given in terms of the behaviour of the original DA's (language acceptance).

For a large family of F-coalgebras, both (i) and (ii) can be captured simultaneously with the help of the categorical notion of *monad*, which generalizes the notion of algebraic theory. The structuring of the state space X can be expressed as a change of functor type from $F(X)$ to $F(T(X))$. In our examples above, both the functors $T_1(X) = \mathcal{P}_\omega(X)$ and $T_2(X) = 1 + X$ are monads, and NDA's and PA's are obtained from DA's by changing the original functor type $D(X)$ into $N(X) = D(T_1(X))$ and $P(X) = D(T_2(X))$. Regarding (ii), one assigns F-semantics to an FT-coalgebra (X, f) by transforming it into an F-coalgebra $(T(X), f^\sharp)$, again using the monad T. In our examples above, the determinization of NDA's and the totalization of PA's consists of the transformation of N- and P-coalgebras (X, f) into D-coalgebras $(T_1(X), f^\sharp)$ and $(T_2(X), f^\sharp)$, respectively.

We shall investigate general conditions on the functor types under which the above constructions can be applied: for one thing, one has to ensure that the FT-coalgebra map $f\colon X \to F(T(X))$ induces a suitable F-coalgebra map $f^\sharp\colon T(X) \to F(T(X))$. Our results will lead to a uniform treatment of all kinds of existing and new variations of automata, that is, FT-coalgebras, by an algebraic structuring of their state space through a monad T. Furthermore, we shall prove a number of general properties that hold in all situations similar to the ones above. For instance, there is the notion of N-behavioural equivalence with which NDA's, being N-coalgebras, come equipped. It coincides with the well-known notion of Park-Milner bisimilarity from process algebra. A general observation is that if two states in an NDA are N-equivalent then they are also D- (that is, language-) equivalent. For PA's, a similar statement holds. One further contribution of this paper is a proof of these statements, once and for all for all FT-coalgebras under consideration.

Coalgebras of type FT were studied in [29, 4, 22]. In [4, 22] the main concern was definitions by coinduction, whereas in [29] a proof principle was also presented. All in all, the present paper can be seen as the understanding of the aforementioned papers from a new perspective, presenting a uniform view on various automata constructions and equivalences.

The structure of the paper is as follows. After preliminaries (Section 1) and the details of the motivating examples above (Section 2), Section 3 presents the general construction as well as many more examples, including the coalgebraic chracterisation of pushdown automata (Section 3.2). In Section 4, a large family of automata (technically: functors) is characterised to which the constructions above can be applied. Section 5 contains the application of the framework in order to recover several interesting equivalences stemming from the world of concurrency, such as failure and ready semantics. Section 6 discusses related work and presents pointers to future work.

This paper is an extended version of [43]. Compared to the conference version, we include the proofs and more examples. More interestingly, the characterisation of pushdown automata coalgebraically (Section 3.2) and the material in Section 5 are original.

1. Background

In this section we introduce the preliminaries on coalgebras and algebras. First, we fix some notation on sets. We will denote sets by capital letters $X, Y, \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is the cartesian product of X and Y (with the usual projection maps π_1 and π_2), $X + Y$ is the disjoint union (with injection maps κ_1 and κ_2) and X^Y is the set of functions $f\colon Y \to X$. The collection of finite subsets of X is denoted by $\mathcal{P}_\omega(X)$, while the collection of full-probability distributions with finite support is $\mathcal{D}_\omega(X) = \{f\colon X \to [0,1] \mid f \text{ finite support and } \sum_{x\in X} f(x) = 1\}$. For a set of letters A, A^* denotes the set of all words over A; ϵ the empty word; and $w_1 \cdot w_2$ (and $w_1 w_2$) the concatenation of words $w_1, w_2 \in A^*$.

1.1. **Coalgebras.** A coalgebra is a pair $(X, f\colon X \to F(X))$, where X is a set of states and $F\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor F, together with the function f, determines the *transition structure* (or dynamics) of the F-coalgebra [37].

An *F-homomorphism* from an F-coalgebra (X, f) to an F-coalgebra (Y, g) is a function $h\colon X \to Y$ preserving the transition structure, *i.e.*, $g \circ h = F(h) \circ f$.

An F-coalgebra (Ω, ω) is said to be *final* if for any F-coalgebra (X, f) there exists a unique F-homomorphism $[\![-]\!]_X \colon X \to \Omega$. All the functors considered in examples in this paper have a final coalgebra.

Let (X, f) and (Y, g) be two F-coalgebras. We say that the states $x \in X$ and $y \in Y$ are *behaviourally equivalent*, written $x \sim_F y$, if and only if they are mapped into the same element in the final coalgebra, that is $[\![x]\!]_X = [\![y]\!]_Y$.

For weak pullback preserving functors, behavioural equivalence coincides with the usual notion of bisimilarity [37].

1.2. **Algebras.** Monads can be thought of as a generalization of algebraic theories. A *monad* $\mathbf{T} = (T, \mu, \eta)$ is a triple consisting of an endofunctor T on **Set** and two natural transformations: a *unit* $\eta \colon Id \Rightarrow T$ and a *multiplication* $\mu \colon T^2 \Rightarrow T$. They satisfy the following commutative laws

$$\mu \circ \eta_T = id_T = \mu \circ T\eta \quad \text{and} \quad \mu \circ \mu_T = \mu \circ T\mu.$$

Sometimes it is more convenient to represent a monad $\mathbf{T}$, equivalently, as a *Kleisli triple* $(T, (\_)^\sharp, \eta)$ [31], where T assigns a set $T(X)$ to each set X, the unit η assigns a function $\eta_X \colon X \to T(X)$ to each set X, and the extension operation $(\_)^\sharp$ assigns to each $f \colon X \to T(Y)$ a function $f^\sharp \colon T(X) \to T(Y)$, such that,

$$f^\sharp \circ \eta_X = f \qquad (\eta_X)^\sharp = id_{T(X)} \qquad (g^\sharp \circ f)^\sharp = g^\sharp \circ f^\sharp \,,$$

for $g \colon Y \to T(Z)$. Monads are frequently referred to as *computational types* [32]. We list now a few examples. In what follows, $f \colon X \to T(Y)$ and $c \in T(X)$.

Nondeterminism. $T(X) = \mathcal{P}_\omega(X)$; η_X is the singleton map $x \mapsto \{x\}$; $f^\sharp(c) = \bigcup_{x \in c} f(x)$.

Partiality. $T(X) = 1 + X$ where $1 = \{*\}$ represents a terminating (or diverging) computation; η_X is the injection map $\kappa_2 \colon X \to 1 + X$; $f^\sharp(\kappa_1(*)) = \kappa_1(*)$ and $f^\sharp(\kappa_2(x)) = f(x)$.

Further examples of monads include: exceptions ($T(X) = E + X$), side-effects ($T(X) = (S \times X)^S$), interactive output ($T(X) = \mu v.X + (O \times v) \cong O^* \times X$) and full-probability ($T(X) = \mathcal{D}_\omega(X)$). We will use all these monads in our examples and we will define η_X and $f^\sharp$ for each later in Section 3.1.

A $\mathbf{T}$-*algebra* of a monad $\mathbf{T}$ is a pair (X, h) consisting of a set X, called carrier, and a function $h \colon T(X) \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id_X$. A T-homomorphism between two $\mathbf{T}$-algebras (X, h) and (Y, k) is a function $f \colon X \to Y$ such that $f \circ h = k \circ Tf$. $\mathbf{T}$-algebras and their homomorphisms form the so-called *Eilenberg-Moore category* $\mathbf{Set}^\mathbf{T}$. There is a forgetful functor $U^\mathbf{T} \colon \mathbf{Set}^\mathbf{T} \to \mathbf{Set}$ defined by

$$U^\mathbf{T}((X, h)) = X \quad \text{and} \quad U^\mathbf{T}(f \colon (X, h) \to (Y, k)) = f \colon X \to Y \,.$$

The forgetful functor $U^\mathbf{T}$ has left adjoint $X \mapsto (T(X), \mu_X \colon TT(X) \to T(X))$, mapping a set X to its free $\mathbf{T}$-algebra. If $f \colon X \to Y$ with (Y, h) a $\mathbf{T}$-algebra, the unique $\mathbf{T}$-homomorphism $f^\sharp \colon (T(X), \mu_X) \to (Y, h)$ with $f^\sharp \circ \eta_X = f$ is given by

$$f^\sharp \colon T(X) \xrightarrow{Tf} T(Y) \xrightarrow{h} Y \,.$$

The function $f^\sharp \colon (T(X), \mu_X) \to (T(Y), \mu_Y)$ coincides with function extension for a Kleisli triple. For the monad $\mathcal{P}_\omega$ the associated Eilenberg-Moore category is the category of join semi-lattices, whereas for the monad $1 + -$ is the category of pointed sets.

2. Motivating examples

In this section, we introduce two motivating examples. We will present two constructions, the determinization of a non-deterministic automaton and the totalization of a partial automaton, which we will later show to be an instance of the same, more general, construction.

2.1. **Non-deterministic automata.** A deterministic automaton (DA) over the input alphabet A is a pair $(X, \langle o, t\rangle)$, where X is a set of states and $\langle o, t\rangle \colon X \to 2 \times X^A$ is a function with two components: o, the output function, determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each input letter a the next state. DA's are coalgebras for the functor $2 \times Id^A$. The final coalgebra of this functor is $(2^{A^*}, \langle \epsilon, (-)_a\rangle)$ where 2^{A^*} is the set of languages over A and $\langle \epsilon, (-)_a\rangle$, given a language L, determines whether or not the empty word is in the language ($\epsilon(L) = 1$ or $\epsilon(L) = 0$, resp.) and, for each input letter a, returns the *derivative* of L: $L_a = \{w \in A^* \mid aw \in L\}$. From any DA, there is a unique map l into 2^{A^*} which assigns to each state its behaviour (that is, the language that the state recognizes).

$$\begin{array}{ccc} X & \xrightarrow{\quad l \quad} & 2^{A^*} \\ {\scriptstyle \langle o,t\rangle}\downarrow & & \downarrow{\scriptstyle \langle \epsilon,(-)_a\rangle} \\ 2 \times X^A & \xrightarrow[id \times l^A]{} & 2 \times (2^{A^*})^A \end{array}$$

A non-deterministic automaton (NDA) is similar to a DA but the transition function gives a set of next-states for each input letter instead of a single state. Thus, an NDA over the input alphabet A is a pair $(X, \langle o, \delta\rangle)$, where X is a set of states and $\langle o, \delta\rangle \colon X \to 2 \times (\mathcal{P}_\omega(X))^A$ is a pair of functions with o as before and where δ determines for each input letter a a set of possible next states. In order to compute the language recognized by a state x of an NDA $\mathcal{A}$, it is usual to first determinize it, constructing a DA $\mathbf{det}(\mathcal{A})$ where the state space is $\mathcal{P}_\omega(X)$, and then compute the language recognized by the state $\{x\}$ of $\mathbf{det}(\mathcal{A})$. Next, we describe in coalgebraic terms how to construct the automaton $\mathbf{det}(\mathcal{A})$.

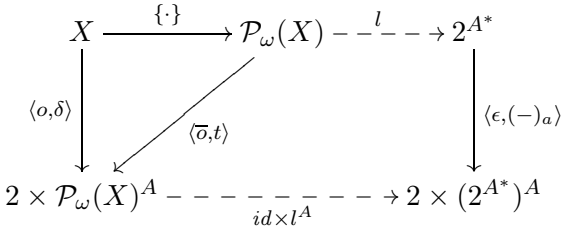
Given an NDA $\mathcal{A} = (X, \langle o, \delta\rangle)$, we construct $\mathbf{det}(\mathcal{A}) = (\mathcal{P}_\omega(X), \langle \overline{o}, t\rangle)$, where, for all $Y \in \mathcal{P}_\omega(X)$, $a \in A$, the functions $\overline{o} \colon \mathcal{P}_\omega(X) \to 2$ and $t \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A$ are

$$\overline{o}(Y) = \begin{cases} 1 & \exists_{y \in Y} o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad t(Y)(a) = \bigcup_{y \in Y} \delta(y)(a).$$

(Observe that these definitions exploit the join-semilattice structures of 2 and $\mathcal{P}_\omega(X)^A$).

The automaton $\mathbf{det}(\mathcal{A})$ is such that the language $l(\{x\})$ recognized by $\{x\}$ is the same as the one recognized by x in the original NDA $\mathcal{A}$ (more generally, the language recognized by state X of $\mathbf{det}(\mathcal{A})$ is the union of the languages recognized by each state x of $\mathcal{A}$).

We summarize the situation above with the following commuting diagram:



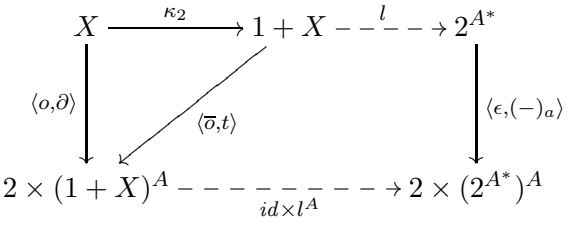
We note that the language semantics of NDA's, presented in the above diagram, can also be obtained as an instance of the abstract definition scheme of λ-coinduction [4, 22].

2.2. **Partial automata.** A partial automaton (PA) over the input alphabet A is a pair $(X, \langle o, \partial\rangle)$ consisting of a set of states X and a pair of functions $\langle o, \partial\rangle \colon X \to 2 \times (1 + X)^A$. Here $o \colon X \to 2$ is the same as with DA. The second function $\partial \colon X \to (1+X)^A$ is a transition function that sends any state $x \in X$ to a function $\partial(x) \colon A \to 1 + X$, which for any input letter $a \in A$ is either undefined (no a-labelled transition takes place) or specifies the next state that is reached. PA's are coalgebras for the functor $2 \times (1 + Id)^A$. Given a PA $\mathcal{A}$, we can construct a total (deterministic) automaton $\mathbf{tot}(\mathcal{A})$ by adding an extra *sink* state to the state space: every undefined a-transition from a state x is then replaced by a a-labelled transition from x to the sink state. More precisely, given a PA $\mathcal{A} = (X, \langle o, \partial\rangle)$, we construct $\mathbf{tot}(\mathcal{A}) = (1 + X, \langle \overline{o}, t\rangle)$, where

$$\begin{array}{ll} \overline{o}(\kappa_1(*)) = 0 & t(\kappa_1(*))(a) = \kappa_1(*) \\ \overline{o}(\kappa_2(x)) = o(x) & t(\kappa_2(x))(a) = \partial(x)(a) \end{array}$$

(Observe that these definitions exploit the pointed-set structures of 2 and $1 + X$).

The language $l(x)$ recognized by a state x will be precisely the language recognized by x in the original partial automaton. Moreover, the new sink state recognizes the empty language. Again we summarize the situation above with the help of following commuting diagram, which illustrates the similarities between both constructions:



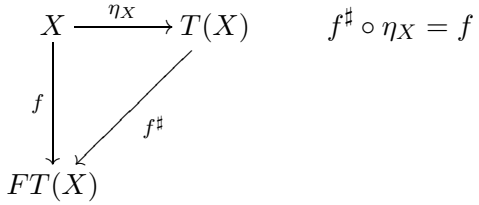
3. Algebraically structured coalgebras

In this section we present a general framework where both motivating examples can be embedded and uniformly studied. We will consider coalgebras for which the functor type FT can be decomposed into a transition type F specifying the relevant dynamics of a system and a monad T providing the state space with an algebraic structure. For simplicity, we fix our base category to be **Set**.

We study coalgebras $f \colon X \to FT(X)$ for a functor F and a monad $\mathbf{T}$ such that $FT(X)$ is a $\mathbf{T}$-algebra, that is $FT(X)$ is the carrier of a $\mathbf{T}$-algebra $(FT(X), h)$. In the motivating

examples, F would be instantiated to $2 \times Id^A$ (in both) and T to $\mathcal{P}_\omega$, for NDAs, and to $1 + -$ for PAs. The condition that $FT(X)$ is a $\mathbf{T}$-algebra would amount to require that $2 \times \mathcal{P}_\omega(X)^A$ is a join-semilattice, for NDAs, and that $2 \times (1 + X)^A$ is a pointed set, for PAs. This is indeed the case, since the set 2 can be regarded both as a join-semilattice ($2 \cong \mathcal{P}_\omega(1)$) or as a pointed set ($2 \cong 1 + 1$) and, moreover, products and exponentials preserve the algebra structure.

The inter-play between the transition type F and the computational type $\mathbf{T}$ (more precisely, the fact that $FT(X)$ is a $\mathbf{T}$-algebra) allows each coalgebra $f\colon X \to FT(X)$ to be extended uniquely to a T-algebra morphism $f^\sharp\colon (T(X), \mu_X) \to (FT(X), h)$ which makes the following diagram commute.



$$f^\sharp \circ \eta_X = f$$

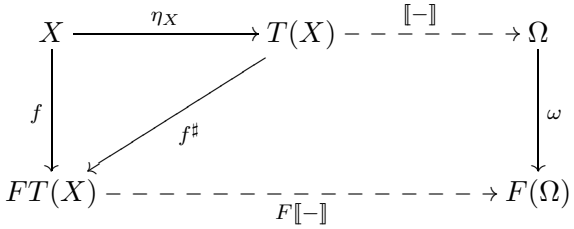
Intuitively, $\eta_X\colon X \to T(X)$ is the inclusion of the state space of the coalgebra $f\colon X \to FT(X)$ into the structured state space $T(X)$, and $f^\sharp\colon T(X) \to FT(X)$ is the extension of the coalgebra f to $T(X)$.

Next, we study the behaviour of a given state or, more generally, we would like to say when two states x_1 and x_2 are equivalent. The obvious choice for an equivalence would be FT-behavioural equivalence. However, this equivalence is not exactly what we are looking for. In the motivating example of non-deterministic automata we wanted two states to be equivalent if they recognize the same language. If we would take the equivalence arising from the functor $2 \times \mathcal{P}_\omega(Id)^A$ we would be distinguishing states that recognize the same language but have difference branching types, as in the following example.

We now define a new equivalence, which *absorbs* the effect of the monad T.

We say that two elements x_1 and x_2 in X are *F-equivalent with respect to a monad* $\mathbf{T}$, written $x_1 \approx_F^T x_2$, if and only if $\eta_X(x_1) \sim_F \eta_X(x_2)$. The equivalence $\sim_F$ is just F-behavioural equivalence for the F-coalgebra $f^\sharp\colon T(X) \to FT(X)$.

If the functor F has a final coalgebra (Ω, ω) , we can capture the semantic equivalence above in the following commuting diagram

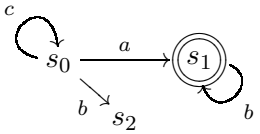


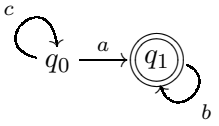
(3.1)

Returning to our first example, two states x_1 and x_2 of an NDA (in which T is instantiated to $\mathcal{P}_\omega$ and F to $2 \times Id^A$) would satisfy $x_1 \approx_F^T x_2$ if and only if they recognize the same language (recall that the final coalgebra of the functor $2 \times Id^A$ is 2^{A^*}).

It is also interesting to remark the difference between the two equivalences in the case of partial automata. The coalgebraic semantics of PAs [39] is given in terms of pairs of prefix-closed languages $\langle V, W\rangle$ where V contains the words that are accepted (that is, are the label of a path leading to a final state) and W contains all words that label any path (that is all that are in V plus the words labeling paths leading to non-final states). We describe V and W in the following two examples, for the states s_0 and q_0:

$W = c^* + c^*b + c^*ab^*$
$V = c^*ab^*$





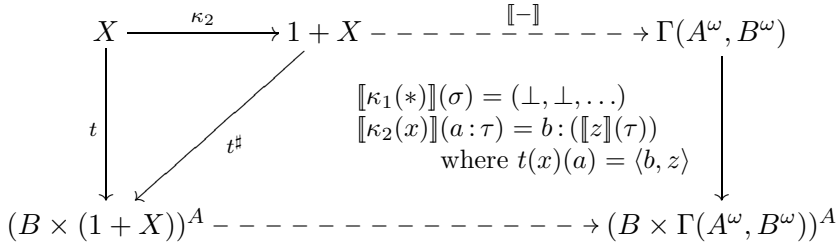
$W = c^* + c^*ab^*$
$V = c^*ab^*$

Thus, the states s_0 and q_0 would be distinguished by FT-equivalence (for $F = 2 \times Id^A$ and $T = 1 + -$) but they are equivalent with respect to the monad $1 + -$, $s_0 \approx_F^T q_0$, since they accept the same language.

We will show in Section 4 that the equivalence $\sim_{FT}$ is always contained in $\approx_F^T$.

3.1. **Examples.** In this section we show more examples of applications of the framework above.

3.1.1. *Partial Mealy machines.* A partial Mealy machine is a set of states X together with a function $t\colon X \to (B \times (1 + X))^A$, where A is a set of inputs and B is a set of output values. We assume that B has a distinguished element $\bot \in B$. For each state x and for each input a the automaton produces an output value and either terminates or continues to a next state. Applying the framework above we will be *totalizing* the automaton, similarly to what happened in the example of partial automata, by adding an extra state to the state space which will act as a sink state. The behaviour of the totalized automaton is given by the set of causal functions from A^ω (infinite sequences of A) to B^ω, which we denote by $\Gamma(A^\omega, B^\omega)$ [38]. A function $f\colon A^\omega \to B^\omega$ is causal if, for $\sigma \in A^\omega$, the n-th value of the output stream $f(\sigma)$ depends only on the first n values of the input stream σ. In the diagram below, we define the final map $[\![-]\!]\colon 1 + X \to \Gamma(A^\omega, B^\omega)$:



Here $* \in 1$, $x \in X$, $a \in A$, $b \in B$, $\sigma \in A^\omega$, $z \in 1 + X$, and $a{:}\tau$ denotes the prefixing of the stream $\tau \in A^\omega$ with the element a.

3.1.2. *Structured Moore automata.* In the following examples we look at the functor

$$F(X) = T(B) \times X^A$$

for arbitrary sets A and B and an arbitrary monad $\mathbf{T} = (T, \eta, (-)^\sharp)$. The coalgebras of F represents Moore automata with outputs in $T(B)$ and inputs in A. Since $T(B)$ is a $\mathbf{T}$-algebra, $T(X)^A$ is a $\mathbf{T}$-algebra and the product of $\mathbf{T}$-algebras is still a $\mathbf{T}$-algebra, then $FT(X)$ is a $\mathbf{T}$-algebra. For this reason, the (pair of) functions $o\colon X \to T(B)$ and $t\colon X \to T(X)^A$ lift to a (pair of) functions

$$o^\sharp\colon T(X) \to T(B) \qquad t^\sharp\colon T(X) \to T(X)^A$$

The final coalgebra of F is $T(B)^{A^*}$. We can characterize the final map $[\![-]\!]\colon T(X) \to T(B)^{A^*}$, for all $m \in T(X)$, $a \in A$ and $w \in A^*$, by

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & T(X) \dashrightarrow^{[\![-]\!]} T(B)^{A^*} \\ {\scriptstyle\langle o,t\rangle}\downarrow & \swarrow_{\langle o^\sharp, t^\sharp\rangle} & \begin{array}{l}[\![m]\!](\epsilon) = o^\sharp(m) \\ [\![m]\!](a\cdot w) = [\![t^\sharp(m)(a)]\!](w)\end{array} \quad \downarrow{\scriptstyle\langle \epsilon, (-)_a\rangle} \\ T(B)\times T(X)^A & \dashrightarrow & T(B)\times (T(B)^{A^*})^A \end{array}$$

Below we shall look at various concrete instances of this scheme, for different choices of the monad T.

Moore automata with exceptions. Let E be an arbitrary set, the elements of which we think of as exceptions. We consider the *exception monad* $T(X) = E + X$ which has the function $\eta(x) = \kappa_2(x)$ as its unit. We define the lifting $f^\sharp\colon T(X) \to T(Y)$, for any function $f\colon X \to T(Y)$, by $f^\sharp = [id, f]$.

An FT-coalgebra $\langle o, t\rangle\colon X \to (E + B) \times (E + X)^A$ will associate with every state x an output value (either in B or an exception in E) and, for each input a, a next state or an exception. The behaviour of a state x, given by $[\![\eta(x)]\!]$, will be a formal power series over A with output values in $E + B$; that is, a function from A^* to $E + B$. The final map is defined as follows, for all $e \in E$, $x \in X$, $a \in A$, and $w \in A^*$:

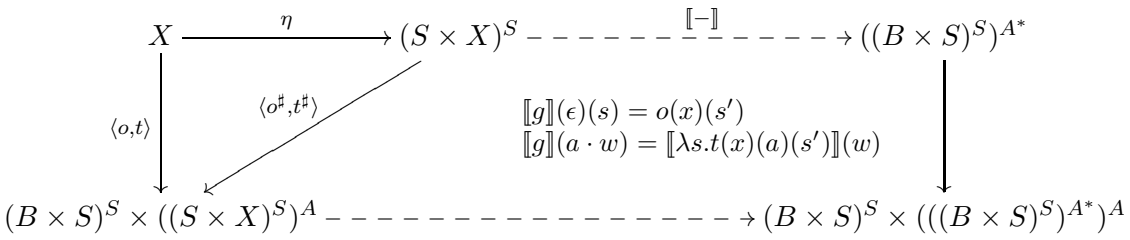
$$\begin{array}{ccc} X & \xrightarrow{\kappa_2} & E + X \dashrightarrow^{[\![-]\!]} (E+B)^{A^*} \\ {\scriptstyle\langle o,t\rangle}\downarrow & \swarrow_{\langle o^\sharp, t^\sharp\rangle} & \begin{array}{l}[\![\kappa_1(e)]\!](w) = \kappa_1(e) \\ [\![\kappa_2(x)]\!](\epsilon) = o(x) \\ [\![\kappa_2(x)]\!](a\cdot w) = [\![t(x)(a)]\!](w)\end{array} \quad \downarrow \\ (E+B)\times (E+X)^A & \dashrightarrow & (E+B)\times ((E+B)^{A^*})^A \end{array}$$

Moore automata with side effects. Let S be an arbitrary set of so-called *side-effects*. We consider the monad $T(X) = (S \times X)^S$, with unit η defined, for all $x \in X$ and $s \in S$, by $\eta(x)(s) = \langle s, x\rangle$. We define the lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ by $f^\sharp(g)(s) = f(x)(s')$, for any $g \in T(X)$ and $s \in S$, and with $g(s) = \langle s', x\rangle$.

Consider an FT-coalgebra $\langle o, t\rangle\colon X \to (B \times S)^S \times ((S \times X)^S)^A$ and let us explain the intuition behind this type of automaton type. The set $S \times X$ can be interpreted as the configurations of the automaton, where S contains information about the state of the system and X about the control of the system. Using the isomorphism $X \to (S \times B)^S \cong$

$S \times X \to S \times B$, we can think of $o\colon X \to (S \times B)^S$ as a function that for each configuration in $S \times X$ provides an output in B and the new state of the system in S. The transition function $t\colon X \to ((S \times X)^S)^A$ gives a new configuration for each input letter and current configuration, using again the fact that $X \to ((S \times X)^S)^A \cong S \times X \to (S \times X)^A$. In all of this, a concrete instance of the set of side-effects could be, for example, the set $S = V^L$ of functions associating memory locations to values.

The behaviour of a state $x \in X$ will be given by $[\![\eta(x)]\!]$, where the final mapping is as follows. For all $g \in (S \times X)^S$, $s \in S$, $a \in A$ and $w \in A^*$, and with $g(s) = \langle s', x\rangle$, we have



Moore automata with interactive output. Let O be an arbitrary set of *outputs*. Consider the interactive output monad defined by the functor $T(X) = \mu v.X + (O \times v) \cong O^* \times X$ together with the natural transformation $\eta_X = \lambda x \in X.\langle \epsilon, x\rangle$, and for which the lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ is given by $f^\sharp(\langle w, x\rangle) = \langle ww', y\rangle$ with $f(x) = \langle w', y\rangle$. We consider FT-coalgebras

$$\langle o, t\rangle\colon X \to (O^* \times B) \times (O^* \times X)^A$$

For $B = 1$, the above coalgebras coincide with *(total) subsequential transducers* [17]: $o\colon X \to O^*$ is the final output function; $t\colon X \to (O^* \times X)^A$ is the pairing of the output function and the next state-function.

The behaviour of a state x will be given by $[\![\eta(x)]\!] = [\![\langle \epsilon, x\rangle]\!]$, where, for every $\langle w, x\rangle \in O^* \times X$, $[\![\langle w, x\rangle]\!]\colon A^* \to O^*$, is given by

$$[\![\langle w, x\rangle]\!](\epsilon) \quad = \quad w \cdot o(x) \qquad\qquad [\![\langle w, x\rangle]\!](aw_1) \quad = \quad w \cdot ([\![t(x)(a)]\!](w_1))$$

Probabilistic Moore automata. Consider the monad of probability distributions defined, for any set X, by

$$T(X) = \mathcal{D}_\omega(X)$$

Its unit is given by the Dirac distribution, defined for $x, x' \in X$ by

$$\eta(x)(x') = \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases}$$

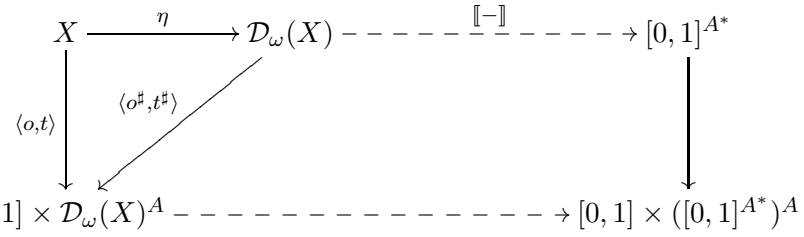
The lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ is given, for any distribution $c \in \mathcal{D}_\omega(X)$ and any $y \in Y$, by

$$f^\sharp(c)(y) = \sum_{d \in \mathcal{D}_\omega(Y)} \left(\sum_{x \in f^{-1}(d)} c(x) \right) \times d(y)$$

We will consider FT-coalgebras

$$\langle o, t\rangle\colon X \to \mathcal{D}_\omega(B) \times \mathcal{D}_\omega(X)^A$$

More specifically, we take $B = 2$ which implies $\mathcal{D}_\omega(2) \cong [0,1]$. For this choice of B, the above FT-coalgebras are precisely the *(Rabin) probabilistic automata* [36]. Each state x has an output value in $o(x) \in [0,1]$ and, for each input a, $t(x)(a)$ is a probability distribution of next states. The behaviour of a state x is given by $[\![\eta(x)]\!] \colon A^* \to [0,1]$, defined below. Intuitively, one can think of $[\![\eta(x)]\!]$ as a probabilistic language: each word is associated with a value $p \in [0,1]$. The final mapping



is given, for any $d \in \mathcal{D}_\omega(X)$, $x \in X$, $a \in A$, and $w \in A^*$, by

$$\begin{aligned} [\![d]\!](\epsilon) &= \sum_{b \in [0,1]} \Big(\sum_{o(x)=b} d(x) \Big) \times b \\ [\![d]\!](aw) &= [\![\lambda x'. \sum_{c \in \mathcal{D}_\omega(X)} (\textstyle\sum_{b=t(x)(a)} d(x)) \times c(x')]\!](w) \end{aligned}$$

It is worth noting that this exactly captures the semantics of [36], while the ordinary $\sim_{FT}$ coincides with *probabilistic bisimilarity* of [28]. Moreover $\approx_F^T$ coincides with the trace semantics of probabilistic transition systems defined in [19] (see Section 7.2 of [23]).

3.2. **Pushdown automata, coalgebraically.** Recursive functions in a computer program lead naturally to a stack of recursive function calls during the execution of the program. In this section, we provide a coalgebraic model of automata equipped with a stack memory. A *pushdown machine* is a tuple (Q, A, B, δ), where Q is set of control locations (states), A is a set of input symbols, B is a set of stack symbols, and δ is finite subset of $Q \times A \times B \times Q \times B^*$, called the set of transition rules. Note that we do not insist on the sets Q, A and B to be finite and consider only *realtime* pushdown machines, i.e. without internal transitions (also called ϵ-transitions) [21]. A *configuration* k of a pushdown machine is a pair $\langle q, \beta \rangle$ denoting the current control state $q \in Q$ and the current content of the stack $\beta \in B^*$. In denoting the stack as a string of stack symbols we assume that the topmost symbol is written first. There is a transition $\langle q, b\beta \rangle \xrightarrow{a} \langle q', \alpha\beta \rangle$ if $\langle q', \alpha \rangle \in \delta(q, a, b)$. A convenient notation is to introduce for any string $w \in A^*$ the transition relation on configurations as the least relation such that

(1) $k \xrightarrow{\epsilon} k$

(2) $k \xrightarrow{aw} k'$ if and only if $k \xrightarrow{a} k''$ and $k'' \xrightarrow{w} k'$.

A *pushdown automaton* (PDA) is a pushdown machine together with an initial configuration k_0 and a set K of accepting configurations. The sets of accepting configurations usually considered are (1) the set $F \times B^*$, where $F \subseteq Q$ is called the set of accepting states, or (2) $Q \times \{\epsilon\}$, but also (3) $F \times \{\epsilon\}$ for $F \subseteq Q$, or (4) $Q \times B'B^*$ for B' a subset of B. A word $w \in A^*$ is said to be accepted by a PDA $(Q, A, B, \delta, k_0, K)$ if $k_0 \xrightarrow{w} k$ for some $k \in K$. A PDA with accepting configurations as in (1) is said to be with accepting states, whereas, when they are as in (2) then the PDA is said to be accepting by empty stack. They both

accept exactly proper context free languages (i.e. context free languages without the empty word) [3].

Computations in a pushdown machine are generally non-deterministic and can cause a change in the control state of the automaton as well as in its stack. For this reason we will model the effects of the computations by means of the so-called *non-deterministic side-effect* monad [5]. For a set of states S, let T be the functor $\mathcal{P}_\omega(- \times S)^S$. It is a monad when equipped with the unit $\eta_X \colon X \to T(X)$, defined by $\eta(x)(s) = \{\langle x, s\rangle\}$, and the multiplication $\mu_X \colon T(T(X)) \to T(X)$ given by

$$\mu_X(k)(s) = \bigcup_{\langle c,s'\rangle \in k(s)} c(s')$$

Note that, for a function $f \colon X \to T(Y)$, the extension $f^\sharp \colon T(X) \to T(Y)$ is defined by

$$f^\sharp(c)(s) = \bigcup_{\langle x',s'\rangle \in c(s)} f(x')(s')\,.$$

Examples of algebras for this monad are $T(1) = \mathcal{P}_\omega(S)^S$ and 2^S. The latter can in fact be obtained as a quotient of the former by equating those functions $k_1, k_2 \colon S \to \mathcal{P}_\omega(S)$ such that for all $s \in S$, $k_1(s) = \emptyset$ if and only if $k_2(s) = \emptyset$.

Every pushdown machine (Q, A, B, δ) together with a set of accepting configurations K induces a function $\langle o, t\rangle \colon Q \to FTQ$ where F is the functor $2^{B^*} \times id^A$ and T is the monad defined above specialized for $S = B^*$ (intuitively, side effects in a pushdown machine are changes in its stack). The functions $o \colon Q \to 2^{B^*}$ and $t \colon Q \to \mathcal{P}_\omega(Q \times B^*)^{B^* A}$ are defined as

$$\begin{array}{lcl} o(q)(\beta) & = & 1 \text{ if and only if } \langle q, \beta\rangle \in K \\ t(q)(a)(\epsilon) & = & \emptyset \\ t(q)(a)(b\beta) & = & \{\langle q', \alpha\beta\rangle \mid \langle q', \alpha\rangle \in \delta(q, a, b)\} \end{array}$$

The transition function t describes the steps between PDA configurations and it is specified in terms of the transition instructions δ of the original machine.

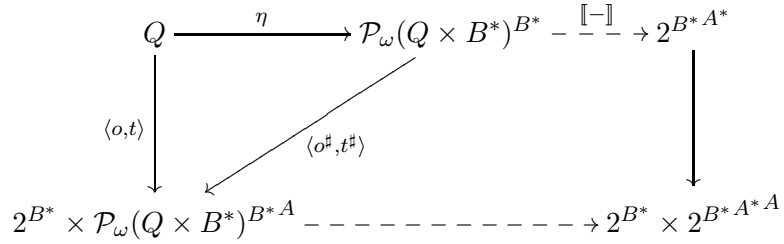
From the above is clear that not every function $\langle o, t\rangle \colon Q \to FTQ$ defines a pushdown machine with accepting configurations, as, for example, $t(q)$ may depend on the whole stack β and not just on the top element b. Therefore we restrict our attention to consider functions $\langle o, t\rangle \colon Q \to FTQ$ such that

(1) $t(q)(a)(\epsilon) = \emptyset$
(2) $t(q)(a)(b\beta) = \{\langle q', \alpha\beta\rangle \mid \langle q', \alpha\rangle \in t(q)(a)(b)\}$,

Every $\langle o, t\rangle$ satisfying (1) and (2) above defines the pushdown machine (Q, A, B, δ) with $\delta(q, a, b) = t(q)(a)(b)$ and with accepting configuration $K = \{\langle q, \beta\rangle | o(q)(\beta) = 1\}$. The first condition is asserting that a machine is in a deadlock configuration when the stack is empty, while the last condition ensures that transition steps depend only on the control state and the top element of the stack. For this reason we will write $q \xrightarrow{a,b|\alpha} q'$ for $\langle q', \alpha\beta\rangle \in t(q)(a)(b)$ indicating that the pushdown machine in the state q by reading an input symbol a and popping b off the stack, can move to a control state q' pushing the string $\alpha \in B^*$ on the current stack (here denoted by β).

Similarly to what we have shown in the examples of structured Moore automata, for every function $\langle o, t\rangle \colon Q \to FTQ$ there is a unique F-coalgebra map $[\![-]\!] \colon T(Q) \to 2^{B^* A^*}$, which is also a T-algebra homomorphism. It is defined for all $c \in \mathcal{P}_\omega(Q \times B^*)^{B^*}$ and $\beta \in B^*$

as

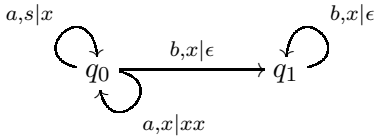


$$[\![\eta(q)]\!](\epsilon) = o(q)$$
$$[\![\eta(q)]\!](aw) = [\![\lambda\beta.t(q)(a)(\beta)]\!](w)$$
$$[\![c]\!](\beta) = \bigcup_{\langle q,\alpha\rangle \in c(\beta)} [\![\eta(q)]\!](\alpha)\,.$$

We then have that a word $w \in A^*$ is *accepted* by the PDA $(Q, A, B, \delta, k_0, K)$ with $k_0 = \langle q, \beta\rangle$ if and only if $[\![\eta(q)]\!](w)(\beta) = 1$.

The above definition implies that for a given word $w \in A^*$ we can decide if it is accepted by $\langle o, t\rangle \colon Q \to FTQ$ from an initial configuration $k_0 = \langle q, \beta\rangle$ in exactly $|w|$ steps (assuming there is a procedure to decide whether $o(q)(\beta) = 1$). As a consequence, we cannot use structured Moore automata to model Turing machines, for which the halting problem is undecidable: in general terms, for Turing machines, we would need internal transitions that do not consume input symbols.

We conclude with an example of our construction using a pushdown machine with control states $Q = \{q_0, q_1\}$, over an input alphabet $A = \{a, b\}$ and using stack symbols $B = \{x, s\}$. The transitions rules δ are given below:



We take $K = \{\langle q_0, \epsilon\rangle, \langle q_1, \epsilon\rangle\}$, meaning that $o(q_0)(\epsilon) = 1$, $o(q_1)(\epsilon) = 1$ and $o(q_i)(\beta) = 0$ in all other cases. By considering $k_0 = \langle q_0, s\rangle$ as initial configuration, we then have

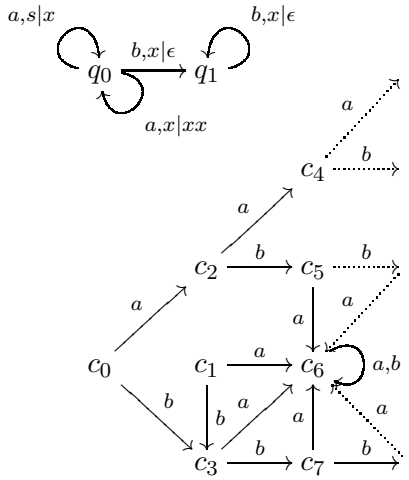
$$[\![\eta(q_0)]\!](\epsilon)(s) = o(q_0)(s) = 0$$

meaning that the empty word is not accepted by the PDA $(Q, A, B, \delta, k_0, K)$. However, the word ab is accepted:

$$\begin{aligned}
[\![\eta(q_0)]\!](ab)(s) &= [\![\lambda\beta.t(q_0)(a)(\beta)]\!](b)(s) \\
&= \bigcup_{\langle p,\beta\rangle \in t(q_0)(a)(s)} [\![\eta(p)]\!](b)(\beta) \\
&= [\![\eta(q_1)]\!](b)(x) \\
&= [\![\lambda\beta.t(q_1)(b)(\beta)]\!](\epsilon)(x) \\
&= \bigcup_{\langle p,\beta\rangle \in t(q_1)(b)(x)} [\![\eta(p)]\!](\epsilon)(\beta) \\
&= [\![\eta(q_1)]\!](\epsilon)(\epsilon) \\
&= o(q_1)(\epsilon) \\
&= 1\,.
\end{aligned}$$

In fact, the language accepted by the above pushdown automaton is $\{a^n b^n \mid n \geq 1\}$. The structured states $c_i \in TQ$, their transitions and their outputs of (part of) the associated Moore automaton are given in Figure 1.

Context-free grammars generating proper languages (i.e. not containing the empty word ϵ) are equivalent to realtime PDA's [11, 13, 42]. Given an input alphabet A, and a set



$$o^\sharp(c_0) = o^\sharp(c_1) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = \epsilon \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c_2) = o^\sharp(c_4) = o^\sharp(c_6) = \lambda\beta.0$$

$$o^\sharp(c_3) = o^\sharp(c_5) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = x \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c_7) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = xxs \\ 0 & \text{otherwise} \end{cases}$$

$$c_0 = \eta(q_0) \qquad c_1 = \eta(q_1)$$

$$c_2 = \lambda\beta. \begin{cases} \{\langle q_0, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_0, xx\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_3 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_4 = \lambda\beta. \begin{cases} \{\langle q_0, xx\beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_0, xxx\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_5 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_1, x\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_6 = \lambda\beta.\emptyset$$

$$c_7 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = xx\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

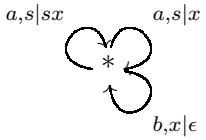
Figure 1: The structured states $c_i \in TQ$, their transitions and their output of (part of) the Moore automaton associated to the PDA $(Q, A, B, \delta, k_0, K)$ where $Q = \{q_0, q_1\}$, $A = \{a, b\}$, $B = \{x, s\}$, δ is depicted on the left top, $k_0 = \langle q_0, s\rangle$ and $K = \{\langle q_0, \epsilon\rangle, \langle q_1, \epsilon\rangle\}$.

of variables B, let $G = (A, B, s, P)$ be a context-free grammar in Greibach normal form [15], i.e. with productions in P of the form $b \to a\alpha$ with $b \in B$, $a \in A$ and $\alpha \in B^*$. We can construct a function $\langle o, t\rangle \colon 1 \to FT1$ (where $1 = \{*\}$) by setting

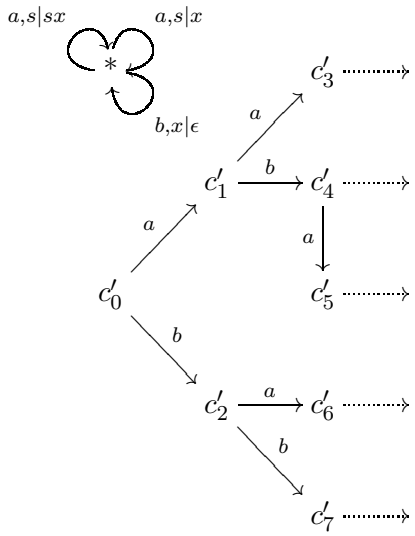
$$o(*)(\beta) = 1 \text{ if and only if } \beta = \epsilon \qquad \text{and} \qquad t(*)(a)(b\beta) = \{\langle *, \alpha\beta\rangle \mid b \to a\alpha \in P\}\,.$$

Clearly this function satisfies conditions (1) and (2) above, and thus, together with the initial configuration $\langle *, s\rangle$ defines a PDA. Furthermore, $[\![\eta(*)]\!](w)(s) = 1$ if and only if there exists a derivation for $w \in A^*$ in the grammar G.

As an example, let us consider the grammar $(\{a, b\}, \{s, x\}, s, P)$ with productions $P = \{s \to asx, s \to ax, x \to b\}$ generating the language $\{a^n b^n \mid n \geq 1\}$. The associated coalgebra $\langle o, t\rangle \colon 1 \to FT1$ is given by



with $o(*)(\beta) = 1$ iff $\beta = \epsilon$



$$c'_0 = \eta(*)$$

$$c'_1 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_2 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_3 = \lambda\beta.\begin{cases} \{\langle *, sxx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_4 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_5 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = ss\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_0) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = \epsilon \\ 0 & \text{otherwise} \end{cases}$$

$$c'_6 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = xs\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_2) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = x \\ 0 & \text{otherwise} \end{cases}$$

$$c'_7 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = xx\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_1) = o^\sharp(c'_3) = o^\sharp(c'_5) = o^\sharp(c'_6) = \lambda\beta.0$$

$$o^\sharp(c'_7) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = xx \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_4) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = s \\ 0 & \text{otherwise} \end{cases}$$

Figure 2: The structured states $c_i \in TQ$, their transitions and their output of (part of) the Moore automaton associated to the PDA $(Q, A, B, \delta, k_0, K)$ where $Q = \{*\}$, $A = \{a, b\}$, $B = \{x, s\}$, δ is depicted on the left top, $k_0 = \langle *, s\rangle$ and $K = \{\langle *, \epsilon\rangle\}$.

Even if the language accepted by the above PDA is the same as the one accepted by the PDA in the previous example (i.e., $[\![\eta(*)]\!](w)(s) = [\![\eta(q_0)]\!](w)(s)$ for all $w \in A^*$), the two associated Moore automaton are not in $\approx_F^T$ (that is $[\![\eta(*)]\!] \neq [\![\eta(q_0)]\!]$). In fact, the Moore automaton associated to the above coalgebra (see below) accepts the string *abab* when starting from the configuration $\langle *, ss\rangle$, while the one in the previous example does not (in symbols, $[\![\eta(*)]\!](abab)(ss) = 1$ while $[\![\eta(q_0)]\!](abab)(ss) = 0$).

The above characterization of context free languages over an alphabet A is different and complementary to the coalgebraic account of context-free languages presented in [44]. The latter, in fact, uses the functor $D(X) = 2 \times X^A$ for deterministic automata (instead of the Moore automata with output in 2^{B^*} above, for B a set of variables), and the idempotent semiring monad $T(X) = \mathcal{P}_\omega((X + A)^*)$ (instead of our side effect monad) to study different

but equivalent ways to present context-free languages: using grammars, behavioural differential equations and generalized regular expressions in which the Kleene star is replaced by a unique fixed point operator.

4. Coalgebras and $\mathbf{T}$-Algebras

In the previous section we presented a framework, parameterized by a functor F and a monad $\mathbf{T}$, in which systems of type FT (that is, FT-coalgebras) can be studied using a novel equivalence $\approx_F^T$ instead of the classical $\sim_{FT}$. The only requirement we imposed was that $FT(X)$ has to be a $\mathbf{T}$-algebra.

In this section, we will present functors F for which the requirement of $FT(X)$ being a $\mathbf{T}$-algebra is guaranteed because they can be *lifted* to a functor F^* on $\mathbf{T}$-algebra. For these functors, the equivalence $\approx_F^T$ coincides with $\sim_{F^*}$. In other words, working on FT-coalgebras in $\mathbf{Set}$ under the novel $\approx_F^T$ equivalence is the same as working on F^*-coalgebras on $\mathbf{T}$-algebras under the ordinary $\sim_{F^*}$ equivalence. Next, we will prove that for this class of functors and an arbitrary monad $\mathbf{T}$ the equivalence $\sim_{FT}$ is contained in $\approx_F^T$. Instantiating this result for our first motivating example of non-deterministic automata will yield the well known fact that bisimilarity implies trace equivalence.

Let $\mathbf{T}$ be a monad. An endofunctor $F^*\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}^{\mathbf{T}}$ is said to be the $\mathbf{T}$-*algebra lifting* of a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ if the following square commutes[1]:

$$\begin{array}{ccc} \mathbf{Set}^{\mathbf{T}} & \xrightarrow{F^*} & \mathbf{Set}^{\mathbf{T}} \\ {\scriptstyle U^{\mathbf{T}}}\downarrow & & \downarrow{\scriptstyle U^{\mathbf{T}}} \\ \mathbf{Set} & \xrightarrow[F]{} & \mathbf{Set} \end{array}$$

If the functor F has a $\mathbf{T}$-algebra lifting F^* then $FT(X)$ is the carrier of the algebra $F^*(T(X), \mu)$. Functors that have a $\mathbf{T}$-algebra lifting are given, for example, by those endofunctors on $\mathbf{Set}$ constructed inductively by the following grammar

$$F ::= Id \mid B \mid F \times F \mid F^A \mid TG$$

where A is an arbitrary set, B is the constant functor mapping every set X to the carrier of a $\mathbf{T}$-algebra (B, h), and G is an arbitrary functor. Since the forgetful functor $U^{\mathbf{T}}\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ creates and preserves limits, both $F_1 \times F_2$ and F^A have a $\mathbf{T}$-algebra lifting if F, F_1, and F_2 have. Finally, TG has a $\mathbf{T}$-algebra lifting for every endofunctor G given by the assignment $(X, h) \mapsto (TGX, \mu_{GX})$. Note that we do not allow taking coproducts in the above grammar, because coproducts of $\mathbf{T}$-algebras are not preserved in general by the forgetful functor $U^{\mathbf{T}}$. Instead, one could resort to extending the grammar with the carrier of the coproduct taken directly in $\mathbf{Set}^{\mathbf{T}}$. For instance, if $\mathbf{T}$ is the (finite) powerset monad, then we could extend the above grammar with the functor $F_1 \oplus F_2 = F_1 + F_2 + \{\top, \bot\}$.

All the functors of the examples in Sections 2 and 3, as well as those in Section 5, can be generated by the above grammar and, therefore, they have a $\mathbf{T}$-algebra lifting.

Now, let F be a functor with a $\mathbf{T}$-algebra lifting and for which a final coalgebra Ω exists. If Ω can be constructed as the limit of the final sequence (for example assuming the functor

[1]This is equivalent to the existence of a distributive law $\lambda\colon TF \Rightarrow FT$ [24].

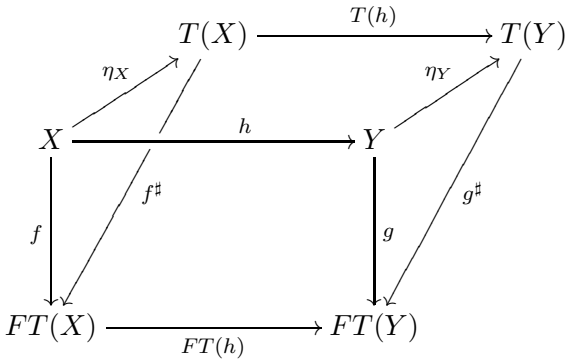
accessible [1]), then, because the forgetful functor $U^{\mathbf{T}}\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ preserves and creates limits, Ω is the carrier of a $\mathbf{T}$-algebra, and it is the final coalgebra of the lifted functor F^*. Further, for any FT-coalgebra $f\colon X \to FT(X)$, the unique F-coalgebra homomorphism $[\![-]\!]$ as in diagram (3.1) is a T-algebra homomorphism between $T(X)$ and Ω. Conversely, the carrier of the final F^*-coalgebra (in $\mathbf{Set}^{\mathbf{T}}$) is the final F-coalgebra (in $\mathbf{Set}$).

Intuitively, the above means that for an accessible functor F with a $\mathbf{T}$-algebra lifting F^*, F^*-equivalence in $\mathbf{Set}^{\mathbf{T}}$ coincides with F-equivalence with respect to $\mathbf{T}$ in $\mathbf{Set}$. The latter equivalence is coarser than the FT-equivalence in $\mathbf{Set}$, as stated in the following theorem.

Theorem 4.1. *Let* $\mathbf{T}$ *be a monad. If* F *is an endofunctor on* $\mathbf{Set}$ *for which a final coalgebra exists and with a* $\mathbf{T}$*-algebra lifting, then* $\sim_{FT}$ *implies* $\approx_F^T$.

Proof. We first show that there exists a functor from the category of FT-coalgebras to the category of F-coalgebras.

This functor maps each FT-coalgebra (X, f) into the F-coalgebra $(T(X), f^\sharp)$ and each FT-homomorphism $h\colon (X, f) \to (Y, g)$ into the F-homomorphism $T(h)\colon (T(X), f^\sharp) \to (T(Y), g^\sharp)$. In order to prove that this is a functor we just have to show that $T(h)$ is an F-homomorphism (i.e., the backward face of the following diagram commutes).



Note that the front face of the above diagram commutes because h is an FT-homomorphism. Also the top face commutes because η is a natural transformation. Thus

$$FT(h) \circ f^\sharp \circ \eta_X = FT(h) \circ f = g \circ h$$

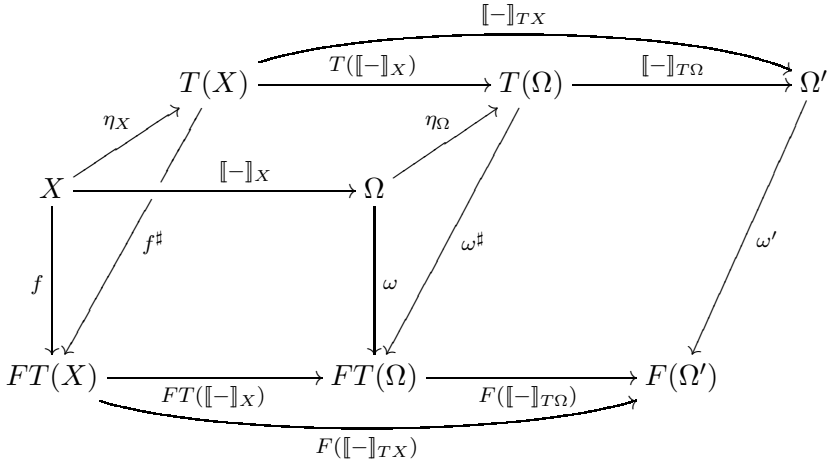
and also

$$g^\sharp \circ T(h) \circ \eta_X = g^\sharp \circ \eta_Y \circ h = g \circ h.$$

Since η is the unit of the adjunction, then there exists a unique $j^\sharp\colon T(X) \to FT(Y)$ in $\mathbf{Set}^{\mathbf{T}}$ such that $g \circ h = j^\sharp \circ \eta_X$. Since both $FT(h) \circ f^\sharp$ and $g^\sharp \circ T(h)$ are (by construction) morphisms in $\mathbf{Set}^{\mathbf{T}}$, then $FT(h) \circ f^\sharp = g^\sharp \circ T(h)$.

Let (X, f) and (Y, g) be two FT-coalgebras and $[\![-]\!]_X$ and $[\![-]\!]_Y$ their morphisms into the final FT-coalgebra (Ω, ω). Let $(T(X), f^\sharp)$, $(T(Y), g^\sharp)$ and $(T(\Omega), \omega^\sharp)$ be the corresponding F-coalgebras and $[\![-]\!]_{TX}$, $[\![-]\!]_{TY}$ and $[\![-]\!]_{T\Omega}$ their morphisms into the final F-coalgebra (Ω', ω').

Since $T([\![-]\!]_X)\colon (T(X), f^\sharp) \to (T(\Omega), \omega^\sharp)$ is an F-homomorphism, then by uniqueness, $[\![-]\!]_{TX} = [\![-]\!]_{T\Omega} \circ T([\![-]\!]_X)$.



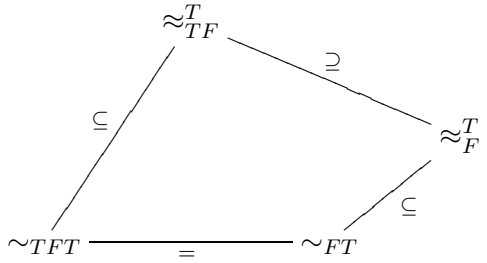
With the same proof, we obtain $[\![-]\!]_{TY} = [\![-]\!]_{T\Omega} \circ T([\![-]\!]_Y)$.

Recall that for all $x \in X$ and $y \in Y$, by definition, $x \sim_{FT} y$ iff $[\![x]\!]_X = [\![y]\!]_Y$ and $x \approx_F^T y$ iff $[\![\eta_X(x)]\!]_{TX} = [\![\eta_Y(y)]\!]_{TY}$.

Suppose that $[\![x]\!]_X = [\![y]\!]_Y$. Then, $T([\![\eta_X(x)]\!]_X) = \eta_\Omega \circ [\![x]\!]_X = \eta_\Omega \circ [\![y]\!]_Y = T([\![\eta_Y(y)]\!]_Y)$ and, finally, $[\![\eta_X(x)]\!]_{TX} = [\![-]\!]_{T\Omega} \circ T([\![\eta_X(x)]\!]_X) = [\![-]\!]_{T\Omega} \circ T([\![\eta_Y(y)]\!]_Y) = [\![\eta_Y(y)]\!]_{TY}$. □

The above theorem instantiates to the well-known facts: for NDA, where $F(X) = 2 \times X^A$ and $T = \mathcal{P}_\omega$, that bisimilarity implies language equivalence; for partial automata, where $F(X) = 2 \times X^A$ and $T = 1 + -$, that equivalence of pairs of languages, consisting of defined paths and accepted words, implies equivalence of accepted words; for probabilistic automata, where $F(X) = [0,1] \times X^A$ and $T = \mathcal{D}_\omega$, that probabilistic bisimilarity implies probabilistic/weighted language equivalence. Note that, in general, the above inclusion is strict.

Remark. Let (X, f) be an FT-coalgebra for a monad $\mathbf{T}$ and a functor F. If $\eta\colon id \Rightarrow T$ is pointwise injective, then $\sim_{FT}$ on the FT-coalgebra (X, f) coincides with $\sim_{TFT}$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$ [37, 4]. If moreover F has a $\mathbf{T}$-algebra lifting then, by the above theorem (on the extended TFT-coalgebra), $\sim_{TFT}$ implies $\approx_{TF}^T$. Combining the two implications, it follows that hat $\sim_{FT}$ on the FT-coalgebra (X, f) implies $\approx_{TF}^T$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$. Finally, under the assumption that F has a $\mathbf{T}$-algebra lifting, we also have that $\approx_F^T$ the FT-coalgebra (X, f) implies $\approx_{TF}^T$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$. This yields the following hierarchy of equivalences.



5. Beyond Bisimilarity and Traces

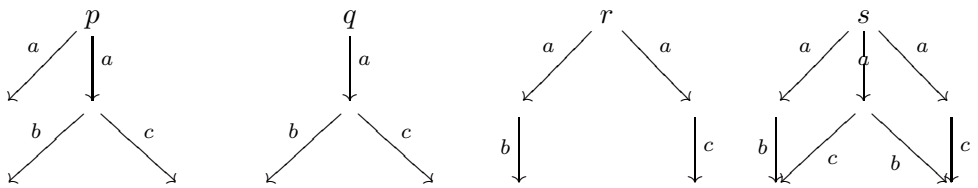
The operational semantics of interactive systems is usually specified by labeled transition systems (LTS's). The denotational semantics is given in terms of behavioural equivalences, which depend the amount of branching structure considered. Bisimilarity (full branching) is sometimes considered too strict, while trace equivalence (no branching) is often considered too coarse. The *linear time / branching time spectrum* [14] shows a taxonomy of many interesting equivalences lying in between bisimilarity and traces.

Labeled transition system are coalgebras for the functor $\mathcal{P}_\omega(Id)^A$ and the coalgebraic equivalence $\sim_{\mathcal{P}_\omega(Id)^A}$ coincides with the standard notion of Park-Milner bisimilarity. In [35], it is shown a coalgebraic characterization of traces semantics (for LTS's) employing Kleisli categories. More recently, [33] have provided a characterization of trace, failure and ready semantics by mean of "behaviour objects". Another coalgebraic approach [26] relies on "test-suite" that, intuitively, are fragments of Hennessy-Milner logic. In this section, we show that (finite) trace equivalence [20], complete trace equivalence [14], failures [9] and ready semantics [34] can be seen as special cases of $\approx_F^T$.

Before introducing these semantics, we fix some notations. A labeled transition system is a pair (X, δ) where X is a set of states and $\delta\colon X \to \mathcal{P}_\omega(X)^A$ is a function assigning to each state $x \in X$ and to each label $a \in A$ a finite set of possible successors states: $x \xrightarrow{a} y$ means that $y \in \delta(x)(a)$. Given a word $w \in A^*$, we write $x \xrightarrow{w} y$ for $x \xrightarrow{a_1} \dots \xrightarrow{a_n} y$ and $w = a_1 \dots a_n$. When $w = \epsilon$, $x \xrightarrow{\epsilon} y$ iff $y = x$. For a function $\varphi \in \mathcal{P}_\omega(X)^A$, $I(\varphi)$ denotes the set of all labels "enabled" by φ, i.e., $\{a \in A \mid \varphi(a) \neq \emptyset\}$, while $\mathit{Fail}(\varphi)$ denotes the set $\{Z \subseteq A \mid Z \cap I(\varphi) = \emptyset\}$.

Let $\langle X, \delta\rangle$ be a LTS and $x \in X$ be a state. A *trace* of x is a word $w \in A^*$ such that $x \xrightarrow{w} y$ for some y. A trace w of x is *complete* if $x \xrightarrow{w} y$ and y stops, i.e., $I(\delta(y)) = \emptyset$. A *failure pair* of x is a pair $\langle w, Z\rangle \in A^* \times \mathcal{P}_\omega(A)$ such that $x \xrightarrow{w} y$ and $Z \in \mathit{Fail}(\delta(y))$. A *ready pair* of x is a pair $\langle w, Z\rangle \in A^* \times \mathcal{P}_\omega(A)$ such that $x \xrightarrow{w} y$ and $Z = I(\delta(y))$. We use $\mathcal{T}(x)$, $\mathcal{CT}(x)$, $\mathcal{F}(x)$ and $\mathcal{R}(x)$ to denote, respectively, the set of all traces, complete traces, failure pairs and ready pairs of x. For $\mathcal{I}$ ranging over $\mathcal{T}, \mathcal{CT}, \mathcal{F}$ and $\mathcal{R}$, two states x and y are $\mathcal{I}$-equivalent iff $\mathcal{I}(x) = \mathcal{I}(y)$.

For an example, consider the following transition systems labeled over $A = \{a, b, c\}$. They are all trace equivalent because their traces are a, ab, ac. The trace a is also complete for p, but not for the others. Only r and s are failure equivalent, since $\langle a, \{bc\}\rangle$ is a failure pair only of p, while $\langle a, \{b\}\rangle$ and $\langle a, \{c\}\rangle$ are failure pairs of p, r and s, but not of q. Finally they are all ready different, since $\langle a, \emptyset\rangle$ is a ready pair only of p, $\langle a, \{b, c\}\rangle$ is a ready pair of q and s but not of r, and $\langle a, \{b\}\rangle$ and $\langle a, \{c\}\rangle$ are ready pairs only of r and s.



We can now show that these equivalences are instances of $\approx_F^T$. We first show ready equivalence in details and then, briefly, the others.

Take $T = \mathcal{P}_\omega$ and $F = \mathcal{P}_\omega(\mathcal{P}_\omega(A)) \times id^A$. For each set X, consider the function $\pi^{\mathcal{R}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ defined for all $\varphi \in \mathcal{P}_\omega(X)^A$ by

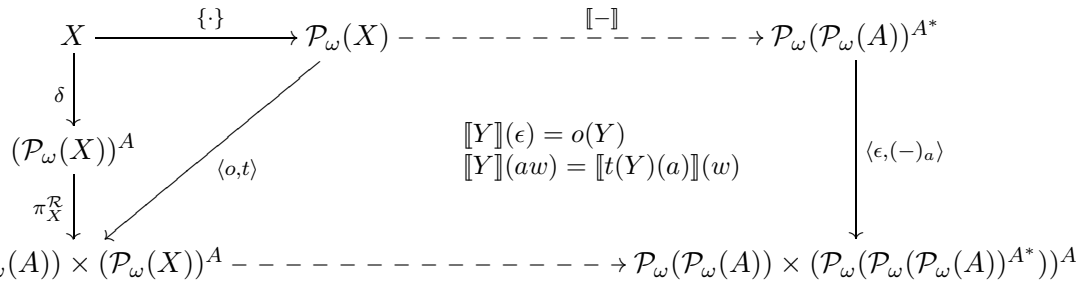
$$\pi^{\mathcal{R}}_X(\varphi) = \langle \{I(\varphi)\}, \varphi \rangle.$$

This function allows to transform each LTS (X, δ) into the FT-coalgebra $(X, \pi^{\mathcal{R}}_X \circ \delta)$. The latter has the same transitions of $\langle X, \delta \rangle$, but each state x is "decorated" with the set $\{I(\varphi)\}$.

Now, by employing the powerset construction, we transform $\langle X, \pi^{\mathcal{R}}_X \circ \delta \rangle$ into the F-coalgebra $(\mathcal{P}_\omega(X), \langle o, t \rangle)$, where, for all $Y \in \mathcal{P}_\omega(X)$, $a \in A$, the functions $o \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))$ and $t \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A$ are

$$o(Y) = \bigcup_{y \in Y} \{I(\delta(y))\} \qquad t(Y)(a) = \bigcup_{y \in Y} \delta(y)(a).$$

The final F-coalgebra is $(\mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}, \langle \epsilon, (-)_a \rangle)$ where $\langle \epsilon, (-)_a \rangle$ is defined as usual.



Summarizing, the final map $[\![-]\!] \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}$ maps each $\{x\}$ into a function assigning to each word w, the set $\{Z \subseteq A \mid x \overset{w}{\to} y \text{ and } Z = I(\delta(y))\}$. In other terms, $Z \in [\![\{x\}]\!](w)$ iff $\langle w, Z \rangle \in \mathcal{R}(x)$.

For the state s depicted above, $[\![\{s\}]\!](\epsilon) = \{\{a\}\}$, $[\![\{s\}]\!](a) = \{\{b\}, \{b, c\}, \{c\}\}$, $[\![\{s\}]\!](ab) = [\![\{s\}]\!](ac) = \{\emptyset\}$ and for all the other words w, $[\![\{s\}]\!](w) = \emptyset$.

The other semantics can be characterized in the same way, by choosing different functors F and different functions $\pi_X \colon \mathcal{P}_\omega(X)^A \to FT$.

For failure semantics, take the same functor as for the ready semantics, that is $F = \mathcal{P}_\omega(\mathcal{P}_\omega(A)) \times id^A$ and a new function $\pi^{\mathcal{F}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ defined $\forall \varphi \in \mathcal{P}_\omega(X)^A$ by

$$\pi^{\mathcal{F}}_X(\varphi) = \langle \mathit{Fail}(\varphi), \varphi \rangle.$$

The FT-coalgebra $(X, \pi^{\mathcal{F}}_X \circ \delta)$ has the same transitions of the LTS $\langle X, \delta \rangle$, but each state x is "decorated" with the set $\mathit{Fail}(\varphi)$.

For both trace and complete trace equivalence, take $F = 2 \times id^A$ (as for NDA). For trace equivalence, $\pi^{\mathcal{T}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ maps $\varphi \in \mathcal{P}_\omega(X)^A$ into $\langle 1, \varphi \rangle$. Intuitively, $(X, \pi^{\mathcal{T}}_X \circ \delta)$ is an NDA where all the states are accepting. For complete traces, $\pi^{\mathcal{CT}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ maps φ in $\langle 1, \varphi \rangle$ if $I(\varphi) = \emptyset$ (and in $\langle 0, \varphi \rangle$ otherwise).

By taking $T = \mathcal{D}_\omega$ instead of $T = \mathcal{P}_\omega$, we hope to be able to characterize probabilistic trace, complete trace, ready and failure as defined in [25].

6. Discussion

In this paper, we lifted the powerset construction on automata to the more general framework of FT-coalgebras. Our results lead to a uniform treatment of several kinds of existing and new variations of automata (that is, FT-coalgebras) by an algebraic structuring of their state space through a monad T. We showed as examples partial Mealy machines, structured Moore automata, nondeterministic, partial and probabilistic automata. Furthermore, we have presented an interesting coalgebraic characterization of pushdown automata and showed how several behavioural equivalences stemming from concurrency theory can be retrieved from the general framework. It is worth mentioning that the framework instantiates to many other examples, among which are *weighted automata* [41]. These are simply structured Moore automata for $B = 1$ and $\mathbf{T} = \mathbb{S}^-_\omega$ (for a semiring $\mathbb{S}$) [16]. It is easy to see that $\sim_{FT}$ coincides with weighted bisimilarity [10], while $\approx^T_F$ coincides with weighted language equivalence [41].

Some of the aforementioned examples can also be coalgebraically characterized in the framework of [19, 18]. There, instead of considering FT-coalgebras on $\mathbf{Set}$ and F^*-coalgebras on $\mathbf{Set}^{\mathbf{T}}$ (the Eilenberg-Moore category), TG-coalgebras on $\mathbf{Set}$ and $\overline{G}$-coalgebras on $\mathbf{Set}_{\mathbf{T}}$ (the *Kleisli* category) are studied. The main theorem of [19] states that under certain assumptions, the initial G-algebra is the final $\overline{G}$-coalgebra that characterizes (generalized) trace equivalence. The exact relationship between these two approaches has been studied in [23] (and, indirectly, it could be deduced from [6] and [27]). It is worth to remark that many of our examples do not fit the framework in [19]: for instance, the exception, the side effect, the full-probability and the interactive output monads do not fulfill their requirements (the first three do not have a bottom element and the latter is not commutative). Moreover, we also note that the example of partial Mealy machines is not purely trace-like, as all the examples in [19].

The idea of using monads for modeling automata with non-determinism, probabilism or side-effects dates back to the "λ-machines" of [2] that, rather than coalgebras, rely on algebras. More precisely, the dynamic of a λ-machine is a morphism $\delta\colon FX \to TX$, where F is a functor and T is a monad (for instance the transitions of T-structured Moore automata are a function $\delta\colon X \times A \to TX$ mapping a state and an input symbol into an element of TX). Analogously to our approach, each λ-machine induces an "implicit λ-machine" having TX as state space. Many examples of this paper (like Moore automata) can be seen as λ-machines, but those systems that are essentially coalgebraic (like Mealy machines) do not fit the framework in [2].

There are several directions for future research. On the one hand, we will try to exploit *F-bisimulations up to T* [29, 30] as a sound and complete proof technique for $\approx^T_F$. On the other hand, we would like to lift many of those coalgebraic tools that have been developed for "branching equivalences" (such as coalgebraic modal logic [12, 40] and (axiomatization for) regular expressions [8]) to work with the "linear equivalences" induced by $\approx^T_F$.

We have pursued further the applications to decorated traces and the challenging modeling of the full linear-time spectrum in a separate paper [7], work which we also plan to extend to probabilistic traces.

References

[1] J. Adámek. Free algebras and automata realization in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

[2] M. Arbib, and E. Manes. Fuzzy machines in a category. *Bull. Austral. Math. Soc.*, 13:169–210, 1975.

[3] J.-M. Autebert, J. Berstel, and L. Boasson. Context-Free Languages and Push-Down Automata. In G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Volume 1, pages 111-174. Springer-Verlag, 1997.

[4] F. Bartels. *On generalized coinduction and probabilistic specification formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[5] N. Benton, J. Hughes, and E. Moggi. Monads and Effects. Course notes for *APPSEM Summer School*, 2000. Available on line at `http://www.disi.unige.it/person/MoggiE/APPSEM00/BHM.ps`.

[6] A. Balan, and A. Kurz. On Coalgebras over Algebras. *Electronic Notes in Theoretical Computer Science*. 264(2): 47-62 (2010)

[7] F. Bonchi, M.M. Bonsangue, G. Caltais, J.J.M.M. Rutten, and A. Silva. Final semantics for decorated traces, In *Proceedings of MFPS*, ENTCS, Elsevier, 2012, to appear.

[8] M.M. Bonsangue, J.J.M.M. Rutten, and A. Silva. An algebra for Kripke polynomial coalgebras. In *Proceedings of 24th Annual IEEE Symposium on Logic In Computer Science (LICS 2009)*, pages 49–58. IEEE Computer Society, 2009.

[9] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, ACM 1984.

[10] P. Buchholz. Bisimulation relations for weighted automata. *Theoretical Computer Science*, 393(1-3):109–123, Elsevier, 2008.

[11] N. Chomsky. Context Free Grammars and Pushdown Storage. *Quarterly Progress Report*, volume 65, MIT Research Laboratory in Electronics, Cambridge, MA, 1962.

[12] C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. *Computer Journal* 54(1):31–41, Oxford University Press, 2011.

[13] R.J. Evey. Application of Pushdown Store Machines. In *Proceedings of the 1963 Fall Joint Computer Conference (AFIPS 1963)*, ACM, 1963.

[14] R.J. van Glabbeek. The Linear Time-Branching Time Spectrum. In E. Best (Ed.), *Proceedings of CONCUR 93*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[15] S. Greibach. A Note on Pushdown Store Automata and Regular Systems. *Proceedings of the American Mathematical Society*, 18:263–268, American Mathematical Society 1967.

[16] H.P. Gumm and T. Schröder. Monoid-labeled transition systems. *Electronic Notes in Theoretical Computer Science*, 44(1):184–203, Elsevier 2001.

[17] H.H. Hansen. Coalgebraising subsequential transducers. *Electronic Notes in Theoretical Computer Science*, 203(5):109–129, 2008.

[18] I. Hasuo. *Tracing Anonymity with Coalgebras*. PhD thesis, Radboud University Nijmegen, 2008.

[19] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007.

[20] C. A. R. Hoare. Communicating Sequential Processes. *Communincation of the ACM.*, 21(8):666–677, ACM, 1978.

[21] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[22] B. Jacobs. Distributive laws for the coinductive solution of recursive equations. *Information and Computation*, 204(4): 561-587, 2006.

[23] B. Jacobs, A. Silva, and A. Sokolova. Trace Semantics via Determinization. To appear in *Proceedings of CMCS 12*, in *Lecture Notes in Computer Science*. Springer, 2012.

[24] P.T. Johnstone. Adjoint lifting theorems for categories of algebras. *Bulletin London Mathematical Society*, 7:294–297, 1975.

[25] C. Jou and S.A. Smolka. Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes. In J. Baeten and J.W. Klop (eds), *proceedings of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 367–383, Springer, 1990.

[26] B. Klin. A coalgebraic approach to process equivalence and a coinduction principle for traces. *Electronic Notes in Theoretical Computer Science*, 106:201–218, 2004.

[27] C. Kissig, and A. Kurz. Generic Trace Logics. In *arXiv:1103.3239v1 [cs.LO]*, 2011.

[28] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
[29] M. Lenisa. From Set-theoretic Coinduction to Coalgebraic Coinduction: some results, some problems. *Electronic Notes in Theoretical Computer Science*, 19:2–22, Elsevier, 1999.
[30] M. Lenisa, J. Power and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electronic Notes in Theoretical Computer Science*, 33:230–260, Elsevier, 2000.
[31] E. Manes. *Algebraic theories. Graduate Texts in Mathematics*, 26, Springer 1976.
[32] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
[33] L. Monteiro. A Coalgebraic Characterization of Behaviours in the Linear Time - Branching Time Spectrum. In *proceedings of the 19th International Workshop on Recent Trends in Algebraic Development Techniques (WADT 2008)*, volume 5486 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2009.
[34] E.-R. Olderog and C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informaticae*, 21(1):9–66, 1986.
[35] J. Power and D. Turi. A Coalgebraic Foundation for Linear Time Semantics. *Electronic Notes in Theoretical Computer Science*, 160:305–29, 1999.
[36] M.O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
[37] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, Elsevier, 2000.
[38] J.J.M.M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. *Electronic Notes in Theoretical Computer Science*, 160:305–319, 2006.
[39] J.J.M.M. Rutten. Coalgebra, concurrency, and control. In R. Boel and G. Stremersch (eds.), *proceedings of the 5th Workshop on Discrete Event Systems (WODES 2000)*, pages 31–38, Kluwer, 2000.
[40] L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoretical Computer Science*, 390(2-3):230–247, Elsevier, 2008.
[41] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
[42] M.P. Schützenberger. On Context Free Languages and Pushdown Automata. *Information and Control*, 6:246-264, 1963.
[43] A. Silva, F. Bonchi, M. Bonsangue and J. Rutten. Generalizing the powerset construction, coalgebraically. In proceedings of *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2010)*, volume 8 of *LIPIcs*, pages 272 – 283, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
[44] J. Winter, M.M. Bonsangue, J.J.M.M. Rutten. Context-Free Languages, Coalgebraically. In A. Corradini, B. Klin, and C. Cirstea, (eds.), *Proceedings of 4th Int. Conference on Algebra and Coalgebra in Computer science (CALCO 2011)*, volume 6859 of *Lecture Notes in Computer Science*, pages 359-376, Springer, 2011.